|  |  |  |
| --- | --- | --- |
|  |  |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий (ИТ)

Кафедра Инструментального и прикладного программного обеспечения

(ИиППО)

**ОТЧЕТ О ПРАКТИЧЕСКОЙ РАБОТЕ №11**

**по дисциплине**

«Технологии обработки транзакций клиент-серверных приложений»

Выполнил студент группы ИКБО-20-21 Сидоров С.Д.

Принял Маличенко С.В.

Москва 2024**ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ**

1. Функция empapi.run, написанная на языке Python, принимает параметр типа jsonb. Добавьте трансформацию, чтобы избежать преобразований в текстовый вид и обратно.
2. Отправляйте пользователю, совершившему покупку, письмо-подтверждение с указанием суммы. Для отправки пользуйтесь уже готовой функцией public.sendmail (посмотрите ее определение) или напишите свою. Посылать письмо внутри транзакции покупки неправильно: транзакция может быть оборвана по какой-либо причине, а письмо уже уйдет. Воспользуйтесь механизмом фоновых заданий: в транзакции добавляйте задание на отправку письма. В таком случае оно будет отправлено, только если транзакция завершится успешно.
3. По возможности, повторите задание на языке Java с использованием технологии ORM.

**РЕФЕРАТ**

Отчёт 26 страниц, 8 рисунков, 9 листингов, 5 источников, 1 приложения.

POSTGRESQL, ДОВЕРЕННЫЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ, ТРАНЗАКЦИИ, СЕРВЕРНОЕ ПРОГРАММИРОВАНИЕ, SQL

Объектом разработки является база данных.

Цель работы – изучение и применение концепции транзакций в контексте баз данных с использованием языков программирования.

В процессе работы производилось исследование концепции транзакций в контексте баз данных с использованием языков программирования.

Результатом являются сведения о концепции транзакций в контексте баз данных с использованием языков программирования.

**ВВЕДЕНИЕ**

[ВВЕДЕНИЕ 6](#_Toc22465)

[1. ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ 7](#_Toc2841)

[1.1 Выполнение задания 1 7](#_Toc15253)

[1.2 Выполнение задания 2 9](#_Toc8457)

[1.3 Выполнение задания 3 14](#_Toc18599)

[ЗАКЛЮЧЕНИЕ 20](#_Toc21455)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 21](#_Toc20094)

[ПРИЛОЖЕНИЕ А 22](#_Toc25451)

**ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ**

БД – база данных;

СУБД – система управления базами данных.

**ВВЕДЕНИЕ**

В современном мире, где данные играют важнейшую роль в различных сферах, от бизнеса до повседневной жизни, вопрос вариативности написания кода на стороне сервера является одним из ключевых и влияет на возможности взаимодействия с системой и хранимыми данными. Для этого в PostgreSQL используются различные языки программирования, используемые для написания функций, процедур, триггеров и т.д.

Цель данной практической работы состоит в изучении концепции транзакций в контексте баз данных и применении их на практике с использованием языков программирования.

1. **ВЫПОЛНЕНИЕ ПРАКТИЧЕСКОЙ РАБОТ****Ы**
   1. **Выполнение задания 1**

Для выполнения задания 1 была изменена функций empapi.run, получившая новое название tmprun, и добавлена строка “TRANSFORM FOR TYPE jsonb”, позволяющая не конвертировать в ручную text в jsonb. Обновленный код функции представлен в листинге 1.

Листинг 1 - Код функции tmprun

CREATE OR REPLACE FUNCTION tmprun(func text, params jsonb)

RETURNS text

TRANSFORM FOR TYPE jsonb

AS $python$

p = plpy.prepare("SELECT \* FROM " + plpy.quote\_ident(func) + "($1)", ["jsonb"])

r = p.execute([params])

cols = r.colnames()

collen = {col: len(col) for col in cols}

for i in range(len(r)):

for col in cols:

if len( str(r[i][col]) ) > collen[col]:

collen[col] = len( str(r[i][col]) )

res = ""

res += " ".join( [col.center(collen[col]," ") for col in cols]) + "\n"

res += " ".join( ["-"\*collen[col] for col in cols]) + "\n"

for i in range(len(r)):

res += " ".join( [str(r[i][col]).ljust(collen[col]," ") for col in cols]) + "\n"

return res

$python$ LANGUAGE plpython3u VOLATILE;

Для тестирования работоспособности данной функции была создана функция test\_function выводящая данные из запрашиваемой таблицы. Код данной функции представлен в лиситинге 2.

Листинг 2 - Код функции test\_function

CREATE OR REPLACE FUNCTION test\_function(params jsonb)

RETURNS TABLE(id INT, value TEXT)

AS $$

DECLARE

table\_name text;

query\_text text;

BEGIN

table\_name := params ->> 'table\_name';

query\_text := 'SELECT id, value::text FROM ' || quote\_ident(table\_name);

RETURN QUERY EXECUTE query\_text;

END;

$$ LANGUAGE plpgsql;

Результат демонстрирующий корректную работу функции представлен на рисунке 1.

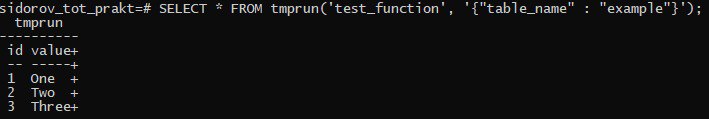


Рисунок 1 - Демонстрация работы функции tmprun

* 1. **Выполнение задания 2**

Для выполнения данного задания использовалась реализация функции sendmail, представленная в листинге 3, которая позволяла пересылать сообщения с использованием smtp google.

Листинг 3 - Код функции sendmail

CREATE OR REPLACE FUNCTION public.sendmail(from\_addr text, to\_addr text, subj text, msg text)

RETURNS void

LANGUAGE plpython3u

AS $function$

import smtplib

server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login('sidorovstasd7@gmail.com', 'uxnyugwremstfqfy')

server.sendmail(

from\_addr,

to\_addr,

"\r\n".join([

"From: %s" % from\_addr,

"To: %s" % to\_addr,

"Content-Type: text/plain; charset=\"UTF-8\"",

"Subject: %s" % subj,

"\r\n%s" % msg

]).encode('utf-8')

)

server.quit()

$function$;

Для реализации отложенной отправки сообщений был использован механизм фоновых заданий, при котором задания записываются в отдельную таблицу и исполняются только по факту успешного завершения транзакции. В результате вызываемая функция внутри транзакции содержит только процесс записи задания в список.

Для демонстрации была создана функция checkout, выполняющаяся для проверки отправляемых заказов, котора представлена в листинге 4.

Листинг 4 - Код функции checkout

CREATE FUNCTION public.checkout(user\_id bigint) returns void

AS $$

BEGIN

PERFORM before\_checkout(user\_id);

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER;

Функция checkout вызывает внутри себя функцию before\_checkout с параметром user\_id, указывающим на пользователе, для которого производится проверка. Функция before\_checkout в свою очередь формирует задание по отправке сообщения с параметрами полученными из заказа. Содержимое функции before\_checkout представлено в листинге 5.

Листинг 5 - Код функции before\_checkout

CREATE OR REPLACE FUNCTION public.before\_checkout(user\_id bigint)

RETURNS void

AS $$

DECLARE

params jsonb;

BEGIN

SELECT jsonb\_build\_object(

'from\_addr', 'sidorovstasd7@gmail.com',

'to\_addr', c.email,

'subj', 'Order Confirmation for User ' || c.name,

'msg', format(

E'Hello %s!\nThank you for your order.\nYour order total is $%s.',

c.name,

total

)

)

INTO params

FROM customer c

JOIN (

SELECT so.customer\_id, SUM(i.total) AS total

FROM sales\_order so

JOIN item i ON so.order\_id = i.order\_id

WHERE so.customer\_id = user\_id

GROUP BY so.customer\_id

) AS order\_totals ON order\_totals.customer\_id = c.customer\_id;

PERFORM public.run\_program(

1,

params,

NULL,

NULL

);

END;

$$ LANGUAGE plpgsql VOLATILE;

Для помещения задания в очередь был произведен вызов функции checkout, что представлено на рисунке 2.

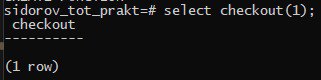


Рисунок 2 - Вызов функции checkout

После этого задание на отправку попало в очередь, что представлено на рисунке 3.

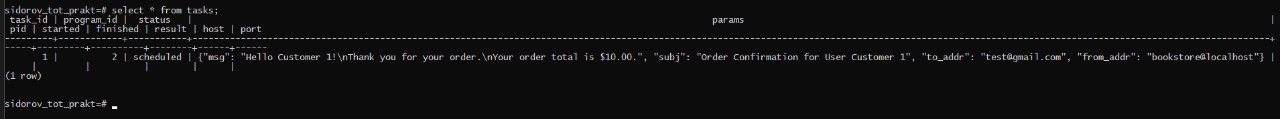


Рисунок 3 - Содержимое таблицы tasks

Для проверки отправки была использована функция execute\_tasks, получающая данные о текущих заданиях и выполняющая их с помощью ранее описанной функции tmprun. В результате после вызова данной функции, который представлен на рисунке 4, было отправлено сообщение.

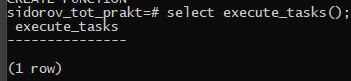


Рисунок 4 - Вызов функции execute\_tasks

Полученное сообщение представлено на рисунке 5.

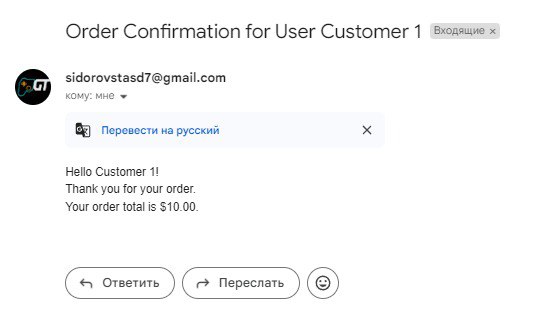


Рисунок 5 - Полученное сообщение

* 1. **Выполнение задания 3**

Для выполнения задания 3 было использовано java spring приложение, воспроизводящее функционал веб-приложения каршеринг. Данное приложение использовало jpa репозитории взаимодействия с postgresql и модели для описания сущностей в приложении, примеры кода репозиториев и моделей представлены в листингах 6 и 7 соответсвенно.

Листинг 6 - Java код репозитория Rents

public interface RentRepo extends JpaRepository<Rent, Integer> {

Optional<List<Rent>> findBySnpassport(Long snpassport);

@Query(value = "SELECT \* FROM rents WHERE snpassport = ?1 AND end\_time IS NULL", nativeQuery = true)

Optional<Rent> findActiveRent(Long snpassport);

Optional<List<Rent>> findByVin(String vin);

}

Листинг 7 - Java код модели Rents

@AllArgsConstructor

@NoArgsConstructor

@Data

@Entity

@Table(name = "rents")

public class Rent {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id\_rent")

private Integer idRent;

@JoinColumn(name = "snpassport")

private Long snpassport;

@Column(name = "duration")

private Integer duration;

@Column(name = "starting\_point")

private String startingPoint;

@Column(name = "start\_time")

private LocalDateTime startTime;

@Column(name = "end\_time")

private LocalDateTime endTime;

@JoinColumn(name = "vin")

private String vin;

@Column(name = "ending\_point")

private String endingPoint;

}

Для рассылки сообщения использовался сервис DefaultEmailService, код которого представлен в листинге 8.

Листинг 8 - Java код класса DefaultEmailService

@Service

public class DefaultEmailService {

@Autowired

public JavaMailSender emailSender;

public void sendEmail(String toEmail, String subject, String body){

SimpleMailMessage message = new SimpleMailMessage();

message.setTo(toEmail);

message.setSubject(subject);

message.setText(body);

emailSender.send(message);

}

}

Для вызова данной службы был изменен код контроллера отвечающего за обработку завершения аренды. Обновленный код представлен в листинге 9.

Листинг 9 - Java код обработчика, отвечающего за завершение аренды

@PostMapping("/close/{id}")

public ResponseEntity<ExecutionResult<Rent>> startRent(@PathVariable Integer id, @RequestBody Rent rent){

ExecutionResult<Rent> result = rentService.closeRent(id, rent);

if (result.getErrorMessage() != null) {

return ResponseEntity.badRequest().body(result);

}

emailService.sendEmail("sidorovstasdw@gmail.com", "TestTOT11", "Testing form java");

return ResponseEntity.ok(result);

}

При таком положении вызова службы отправка сообщения не возможна при не успешном завершении аренды.

Для тестирования работоспособности была создана новая аренда, что представлено на рисунке 6.

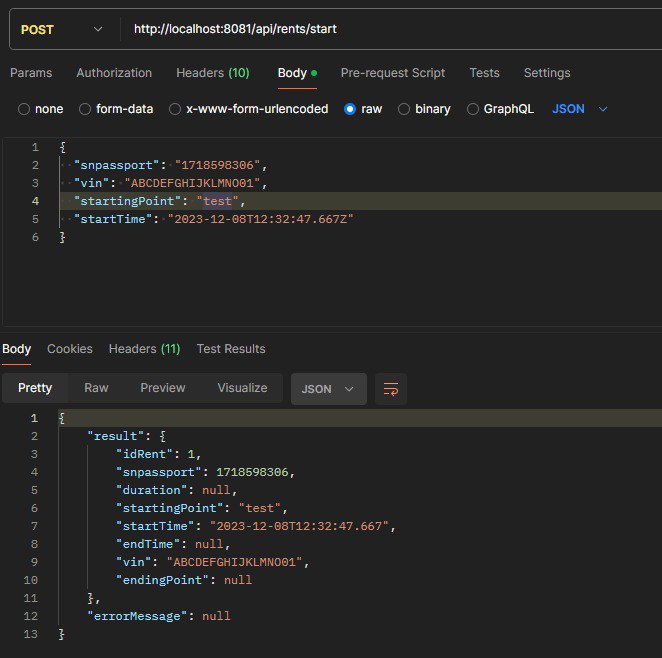


Рисунок 6 - Создание новой аренды

Затем, для отправки сообщения, аренда была завершена, что представлено на рисунке 7.

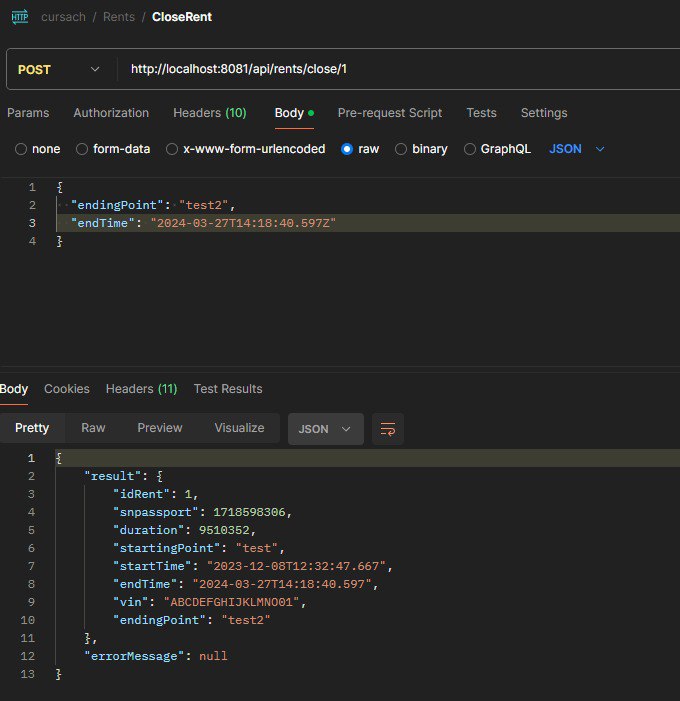


Рисунок 7 - Завершение аренды

В результате на, указанной для получения, почте было получено сообщение, представленное на рисунке 8.

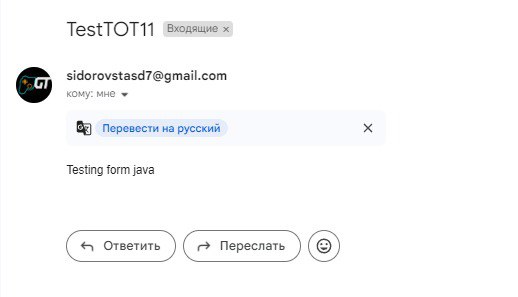


Рисунок 8 - Полученное сообщение

# ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы были получены практические навыки по применению различных языков серверного программирования для решения различных задач на стороне сервера. Также были изучены концепции транзакций в контексте баз данных с применением их на практике с использованием различных языков программирования.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. PostgreSQL: Официальная документация [Электронный ресурс] – URL: https://www.postgresql.org/docs/ (дата обращения: 14.02.2024).
2. PostgreSQL Tutorial for Beginners [Электронный ресурс] – URL: https://www.tutorialspoint.com/postgresql/index.htm (дата обращения: 14.02.2024).
3. PostgreSQL: Википедия [Электронный ресурс] – URL: https://en.wikipedia.org/wiki/PostgreSQL (дата обращения: 14.02.2024).
4. Лузанов П.В. и др. Postgres. Первое знакомство. [Электронный ресурс] – URL: https://postgrespro.ru/education/books/introbook (дата обращения: 22.02.2024)
5. Новиков Б. А. Лекции Основы технологий баз данных. [Электронный ресурс] – URL: https://postgrespro.ru/education/university/dbtech (дата обращения: 22.02.2024)

# ПРИЛОЖЕНИЕ А

**ФРАГМЕНТЫ КОДА РАЗРАБОТАННОГО ПРИЛОЖЕНИЯ**

Листинг 1 – SQL код для выполнения задания 1

CREATE OR REPLACE FUNCTION tmprun(func text, params jsonb)

RETURNS text

TRANSFORM FOR TYPE jsonb

AS $python$

p = plpy.prepare("SELECT \* FROM " + plpy.quote\_ident(func) + "($1)", ["jsonb"])

r = p.execute([params])

cols = r.colnames()

collen = {col: len(col) for col in cols}

for i in range(len(r)):

for col in cols:

if len( str(r[i][col]) ) > collen[col]:

collen[col] = len( str(r[i][col]) )

res = ""

res += " ".join( [col.center(collen[col]," ") for col in cols]) + "\n"

res += " ".join( ["-"\*collen[col] for col in cols]) + "\n"

for i in range(len(r)):

res += " ".join( [str(r[i][col]).ljust(collen[col]," ") for col in cols]) + "\n"

return res

$python$ LANGUAGE plpython3u VOLATILE;

CREATE OR REPLACE FUNCTION test\_function(params jsonb)

RETURNS TABLE(id INT, value TEXT)

AS $$

DECLARE

table\_name text;

query\_text text;

BEGIN

table\_name := params ->> 'table\_name';

query\_text := 'SELECT id, value::text FROM ' || quote\_ident(table\_name);

RETURN QUERY EXECUTE query\_text;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM tmprun('test\_function', '{"table\_name" : "example"}');

Листинг 2 – SQL код для выполнения задания 2 практической работы

CREATE OR REPLACE FUNCTION public.sendmail(from\_addr text, to\_addr text, subj text, msg text)

RETURNS void

LANGUAGE plpython3u

AS $function$

import smtplib

server = smtplib.SMTP('smtp.gmail.com', 587)

server.starttls()

server.login('sidorovstasd7@gmail.com', 'uxnyugwremstfqfy')

server.sendmail(

from\_addr,

to\_addr,

"\r\n".join([

"From: %s" % from\_addr,

"To: %s" % to\_addr,

"Content-Type: text/plain; charset=\"UTF-8\"",

"Subject: %s" % subj,

"\r\n%s" % msg

]).encode('utf-8')

)

server.quit()

$function$;

create table programs (program\_id SERIAL PRIMARY KEY, name text, func text);

CREATE OR REPLACE FUNCTION public.register\_program(name text, func text)

RETURNS bigint

LANGUAGE sql

AS $function$

INSERT INTO programs(name, func) VALUES (name, func)

RETURNING program\_id;

$function$;

SELECT register\_program('Send a mail', 'sendmail\_task');

CREATE FUNCTION public.sendmail\_task(params jsonb)

RETURNS text

AS $$

SELECT sendmail(

from\_addr => params->>'from\_addr',

to\_addr => params->>'to\_addr',

subj => params->>'subj',

msg => params->>'msg'

);

SELECT 'OK';

$$ LANGUAGE sql VOLATILE;

Продолжение листинга 2

CREATE TABLE public.tasks (

task\_id SERIAL PRIMARY KEY ,

program\_id bigint NOT NULL,

status text DEFAULT 'scheduled'::text NOT NULL,

params jsonb,

pid integer,

started timestamp with time zone,

finished timestamp with time zone,

result text,

host text,

port text,

CONSTRAINT tasks\_check CHECK ((((host IS NOT NULL) AND (port IS NOT NULL)) OR ((host IS NULL) AND (port IS NULL)))),

CONSTRAINT tasks\_status\_check CHECK ((status = ANY (ARRAY['scheduled'::text, 'running'::text, 'finished'::text, 'error'::text])))

);

CREATE FUNCTION public.run\_program(program\_id bigint, params jsonb DEFAULT NULL::jsonb, host text DEFAULT NULL::text, port text DEFAULT NULL::text) RETURNS bigint

LANGUAGE sql SECURITY DEFINER

AS $$

INSERT INTO tasks(program\_id, status, params, host, port)

VALUES (program\_id, 'scheduled', params, host, port)

RETURNING task\_id;

$$;

CREATE FUNCTION public.checkout(user\_id bigint) returns void

AS $$

BEGIN

PERFORM before\_checkout(user\_id);

END;

$$ LANGUAGE plpgsql VOLATILE SECURITY DEFINER;

Продолжение листинга 2

CREATE OR REPLACE FUNCTION public.before\_checkout(user\_id bigint)

RETURNS void

AS $$

DECLARE

params jsonb;

BEGIN

SELECT jsonb\_build\_object(

'from\_addr', 'sidorovstasd7@gmail.com',

'to\_addr', c.email,

'subj', 'Order Confirmation for User ' || c.name,

'msg', format(

E'Hello %s!\nThank you for your order.\nYour order total is $%s.', c.name,

total

)

)

INTO params

FROM customer c

JOIN (

SELECT so.customer\_id, SUM(i.total) AS total

FROM sales\_order so

JOIN item i ON so.order\_id = i.order\_id

WHERE so.customer\_id = user\_id

GROUP BY so.customer\_id

) AS order\_totals ON order\_totals.customer\_id = c.customer\_id;

PERFORM public.run\_program(

1,

params,

NULL,

NULL

);

END;

$$ LANGUAGE plpgsql VOLATILE;

select checkout(1);

Продолжение листинга 2

CREATE OR REPLACE FUNCTION public.execute\_tasks()

RETURNS VOID

AS $$

DECLARE

task\_record RECORD;

program\_name TEXT;

task\_params JSONB;

BEGIN

FOR task\_record IN SELECT \* FROM tasks WHERE status = 'scheduled' LOOP

SELECT func INTO program\_name FROM programs WHERE program\_id = task\_record.program\_id;

task\_params := task\_record.params;

PERFORM tmprun(program\_name, task\_params);

UPDATE tasks SET status = 'finished', finished = current\_timestamp WHERE task\_id = task\_record.task\_id;

END LOOP;

END;

$$ LANGUAGE plpgsql;

select execute\_tasks();